Machine Learning Engineering for Production (MLOps) Specialization

by DeepLearning.AI

Course Site

Made By: Matias Borghi

SUMMARY COURSES

COURSE 1 Introduction to Machine Learning in Production

COURSE 2 Machine Learning Data Lifecycle in Production

COURSE 3 Machine Learning Modeling Pipelines in Production

COURSE 4 Deploying Machine Learning Models in Production

Course #1 Introduction to Machine Learning in Production

Week 1: Overview of the ML Lifecycle and Deployment

This week covers a quick introduction to machine learning production systems focusing on their requirements and challenges. Next, the week focuses on deploying production systems and what is needed to do so robustly while facing constantly changing data.

Learning Objectives

- **Identify the key components of the ML Lifecycle.**
- **Define "concept drift" as it relates to ML projects.**
- **Differentiate between shadow, canary, and blue-green deployment scenarios in the context of varying degrees of automation.**
- **Compare and contrast the ML modeling iterative cycle with the cycle for deployment of ML products.**
- **List the typical metrics you might track to monitor concept drift.**

Week 2: Select and Train a Model

This week is about model strategies and key challenges in model development. It covers error analysis and strategies to work with different data types. It also addresses how to cope with class imbalance and highly skewed data sets.

Learning Objectives

- **Identify the key challenges in model development.**
- **Describe how performance on a small set of disproportionately important examples may be more crucial than performance on the majority of examples.**
- **Explain how rare classes in your training data can affect performance.**
- **Define three ways of establishing a baseline for your performance.**
- **Define structured vs. unstructured data.**
- **Identify when to consider deployment constraints when choosing a model.**
- **List the steps involved in getting started with ML modeling.**
- **Describe the iterative process for error analysis.**
- **Identify the key factors in deciding what to prioritize when working to improve model accuracy.**

- **Describe methods you might use for data augmentation given audio data vs. image data.**
- **Explain the problems you can have training on a highly skewed dataset.**
- **Identify a use case in which adding more data to your training dataset could actually hurt performance.**
- **Describe the key components of experiment tracking.**

Week 3: Data Definition and Baseline

This week is all about working with different data types and ensuring label consistency for classification problems. This leads to establishing a performance baseline for your model and discussing strategies to improve it given your time and resources constraints.

Learning Objectives

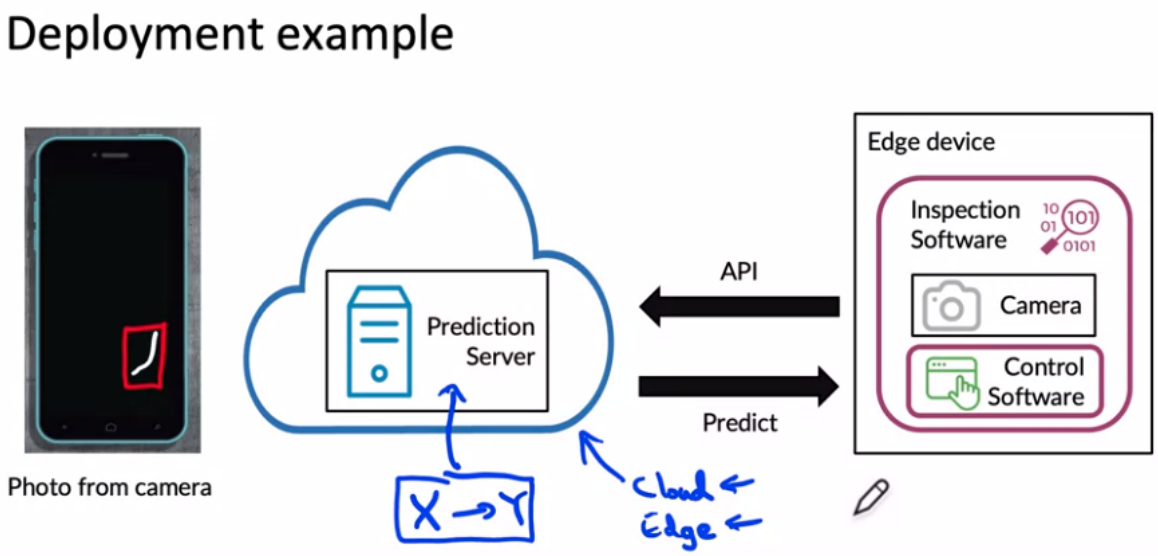
- **List the questions you need to answer in the process of data definition.**
- **Compare and contrast the types of data problems you need to solve for structured vs. unstructured and big vs. small data.**
- **Explain why label consistency is important and how you can improve it**
- **Explain why beating human level performance is not always indicative of success of an ML model.**
- **Make a case for improving human level performance rather than beating it.**
- **Identify how much training data you should gather given time and resource constraints.**
- **Describe the key steps in a data pipeline.**
- **Compare and contrast the proof of concept vs. production phases on an ML project.**
- **Explain the importance of keeping track of data provenance and lineage.**

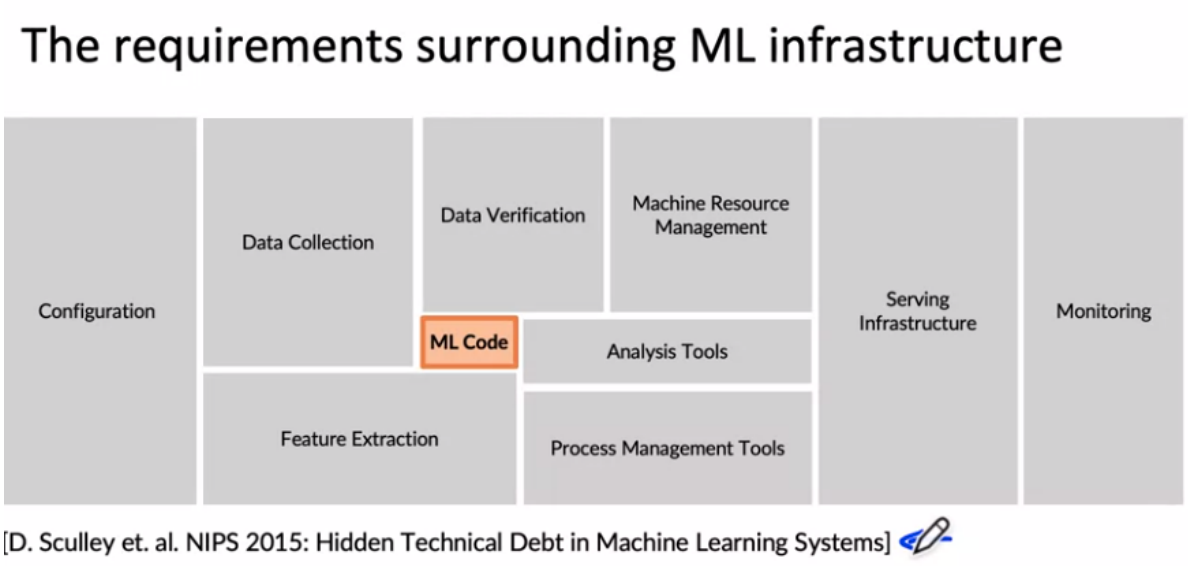
Week 1: Overview of the ML Lifecycle and Deployment

The Machine Learning Project Lifecycle

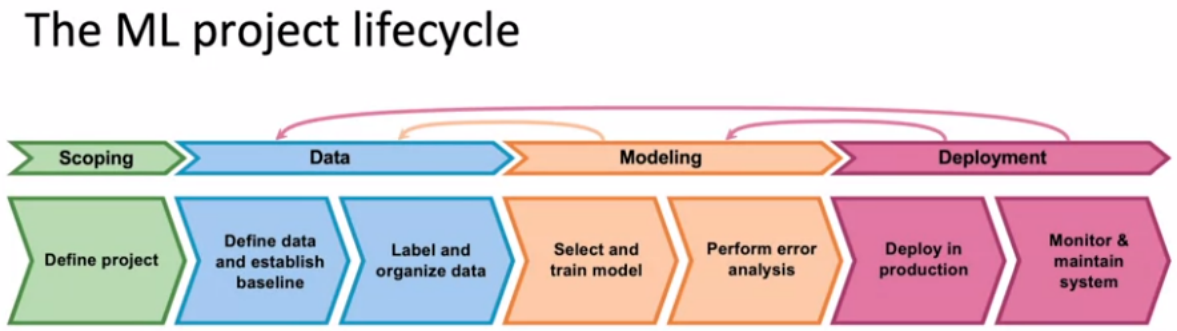
Just because you've trained a learning algorithm that does well on your test set, which is to be celebrated. It's great when you do well when you hold a test set. Unfortunately reaching that milestone doesn't mean you're done. There can still be quite a lot of work and challenges ahead to get a valuable production deployment running.

For example, let's say your training set has images that look like this. There's a good phone on the left, the one in the middle, it has a big scratch across it and you've trained your learning algorithm to recognize that things like this on the left are okay. Meaning that no defects and maybe draw bounding boxes around scratches or other defects that finds and films. When you deploy it in the factory, you may find that the real life production deployment gives you back images like this much darker ones. Because the lighting factory, because the lighting conditions in the factory have changed for some reason compared to the time when the training set was collected. This problem is sometimes called concept drift or data drift.



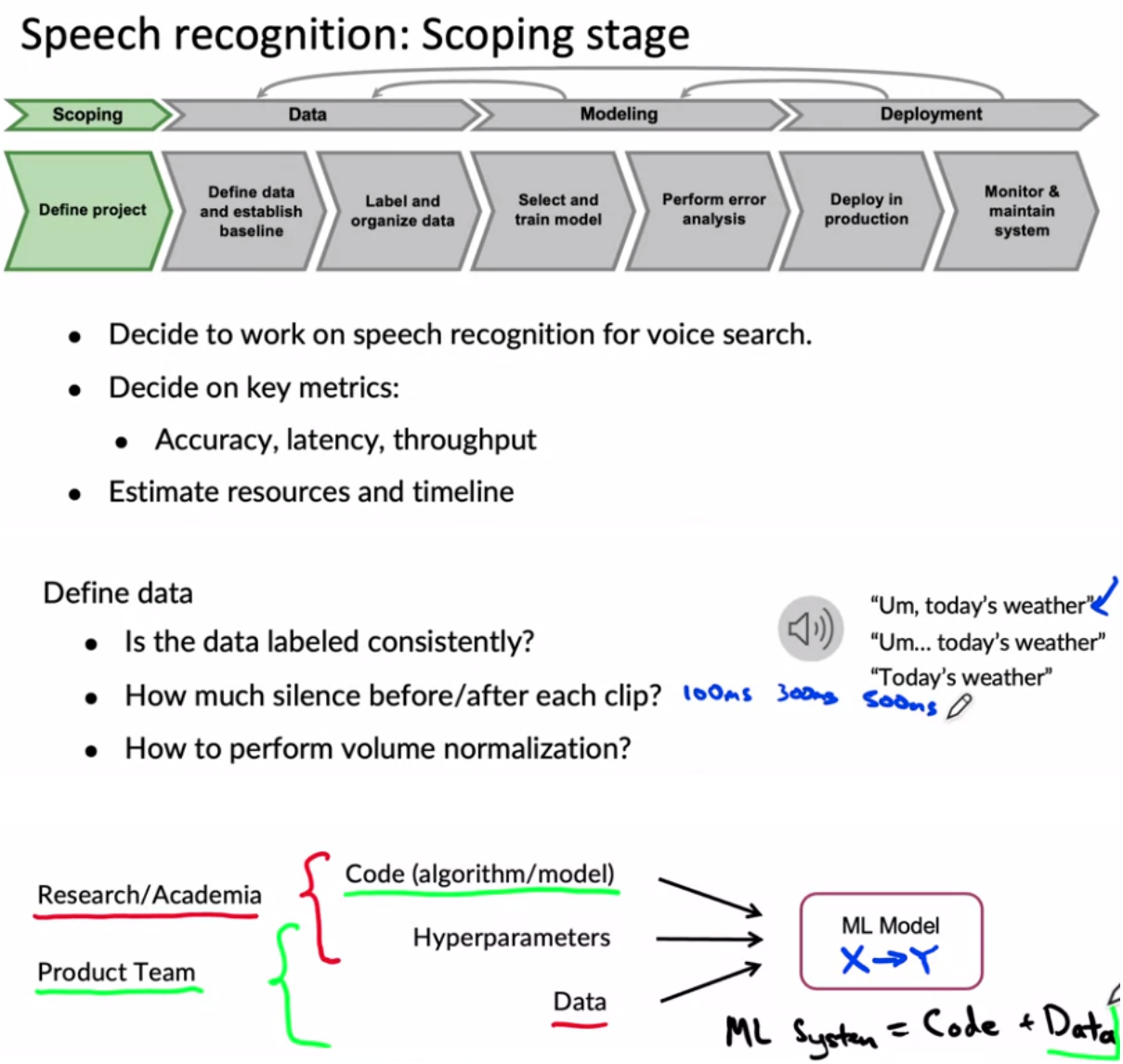


Beyond the machine learning codes there are also many components, especially components for managing the data, such as data collection, data verification, feature extraction.



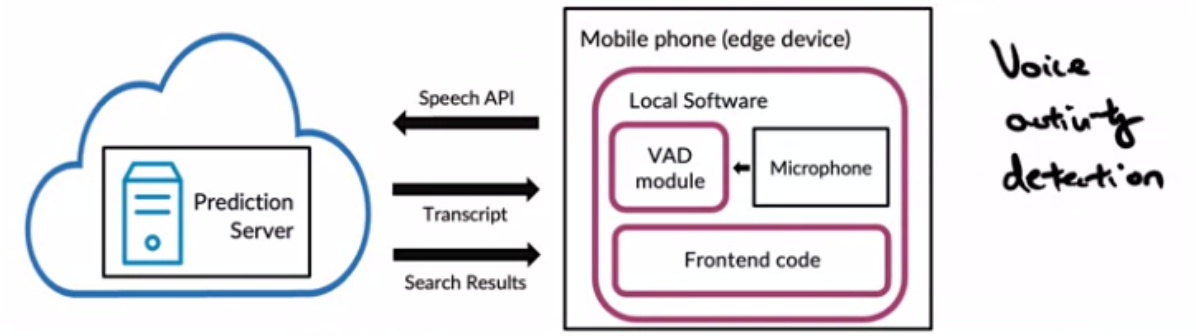
After having chosen the project, you then have to collect data or acquire the data you need for your algorithm.

Now that the system is deployed and is running on live data, and feeding that back into your dataset to then potentially update your data, retrain the model, and so on until you can put an updated model into deployment.

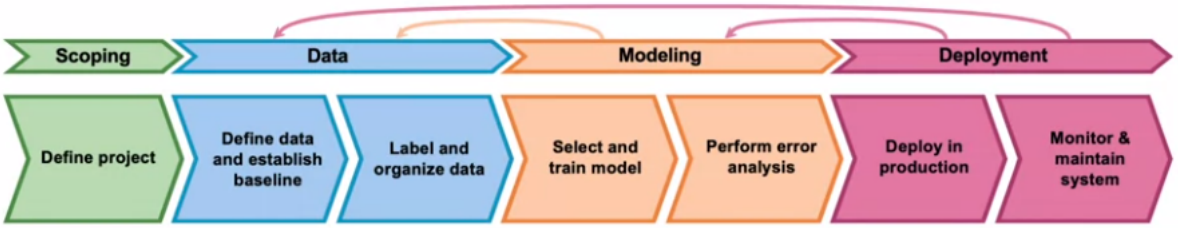


A lot of research work or academic work you tend to hold the data fixed and vary the code and may be varied hyper parameters in order to try to get good performance.

In contrast, for a lot of product teams, if your main goal is to just build and deploy a working valuable machine learning system, can be even more effective to hold the code fixed and to instead focus on optimizing the data and maybe the hyper parameters. In order to get a high performing model, A machine learning system includes both codes and data and also hyper parameters that there maybe a bit easier to optimize than the code or data. And rather than taking a model centric view of trying to optimize the code to your fixed data set for many problems, you can use an open source implementation of something you download of Github and instead just focus on optimizing the data.



Course outline



1. Deployment
2. Modeling
3. Data

Optional: Scoping

MLOps (Machine Learning Operations) is an emerging discipline, and comprises a set of tools and principles to support progress through the ML project lifecycle.

Deployment

Concept drift and Data drift

Speech recognition example

Training set: $x \rightarrow y$

- Purchased data, historical user data with transcripts

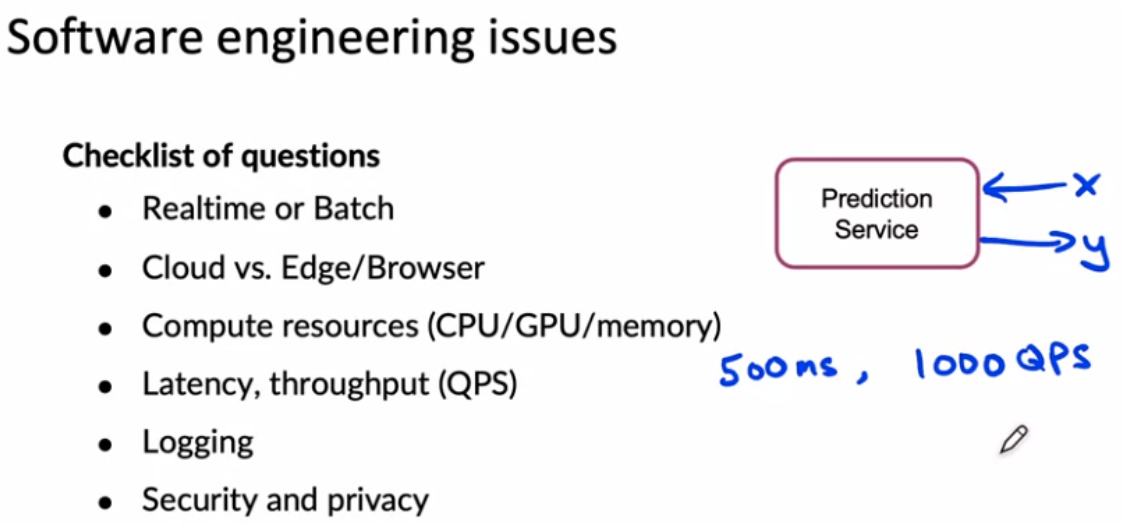
Test set:

- Data from a few months ago

How has the data changed?

Changed: New language/new phone. Gradual change/ Sudden change. Fraud systems failed after covid because people changed their shopping patterns.

Another example of Concept drift, let's say that x is the size of a house, and y is the price of a house, because you're trying to estimate housing prices. If because of inflation or changes in the market, houses may become more expensive over time. The same size house will end up with a higher price. That would be Concept drift. Maybe the size of houses haven't changed, but the price of a given house changes. Whereas data drift would be if, say, people start building larger houses, or start building smaller houses and thus the input distribution of the sizes of houses actually changes over time.



If you save this checklist somewhere, going through this when you're designing your software might help you to make the appropriate software engine choices when implementing your prediction service. To summarize, deploying a system requires two broad sets of tasks: there is writing the software to enable you to deploy the system in production. There is what you need to do to monitor the system performance and to continue to maintain it, especially in the face of concepts drift as well as data drift. One of the things you see when you're building machine learning systems is that the practices for the very first deployments will be quite different compared to when you are updating or maintaining a system that has already previously been deployed.

Deployment patterns

Not just turn on the model and hope for the best

Common deployment cases

1. New product/capability
2. Automate/assist with manual task
3. Replace previous ML system

Key ideas:

- Gradual ramp up with monitoring
- Rollback

1. Example: new recognition system. Pattern: Start with low traffic and then catch up.
2. Already done by a person and we want to automate that task. Example: people in a factory checking subtraction.

Rollback: if algorithm is not working is nice if we can go back to a previous working version.

Visual inspection example

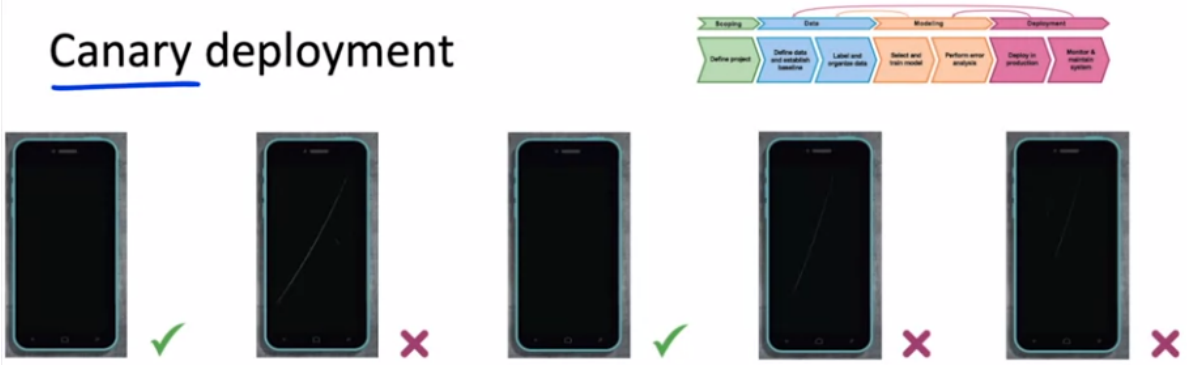
shadow mode

ML system shadows the human and runs in parallel.

ML system's output not used for any decisions during this phase.

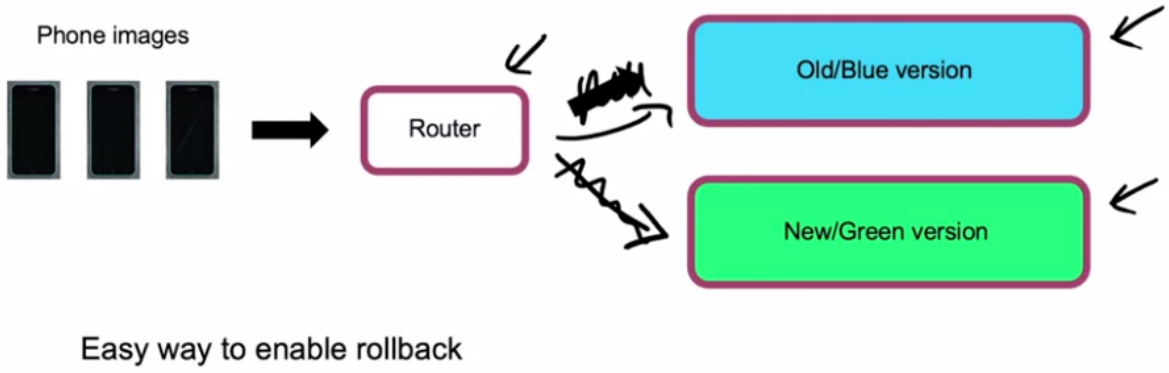
Canary: spot early on problems instead of when we have fully deployed.

Canary deployment



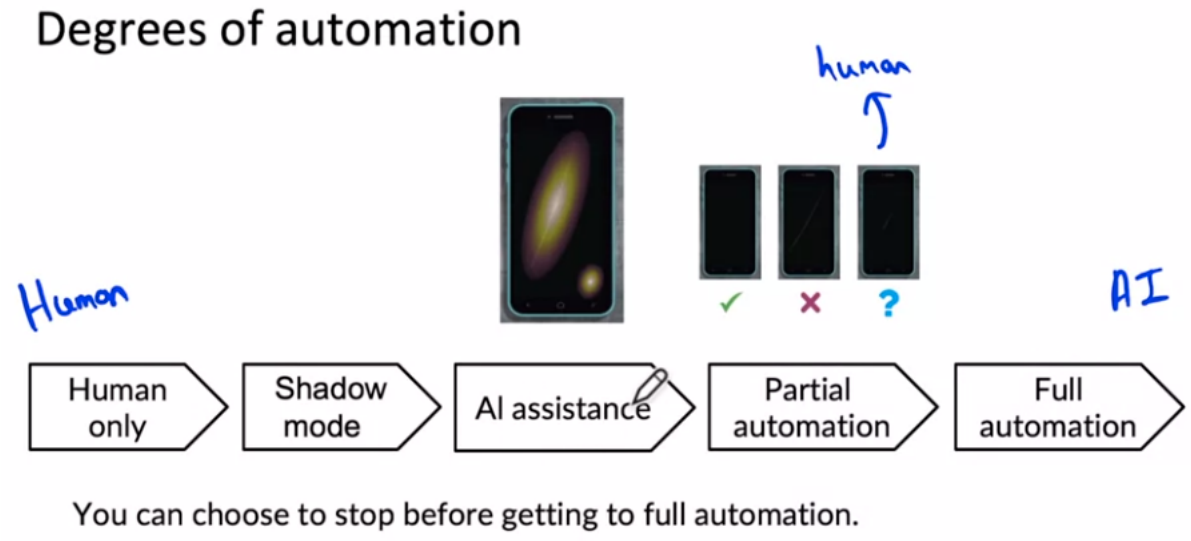
- Roll out to small fraction (say 5%) of traffic initially.
- Monitor system and ramp up traffic gradually.

Blue green deployment

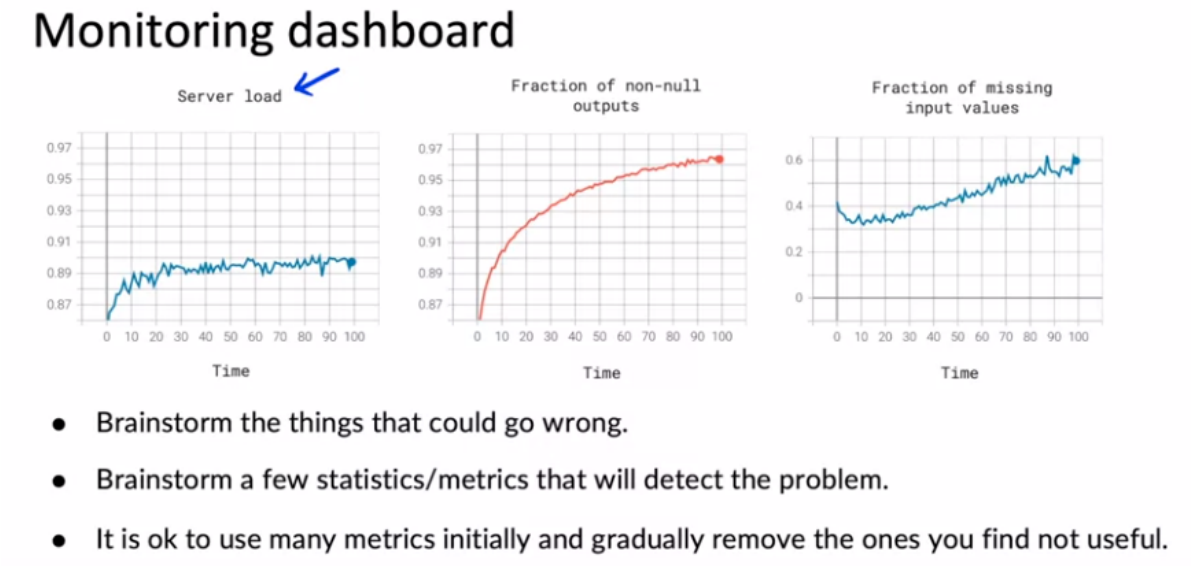


Easy way to enable rollback

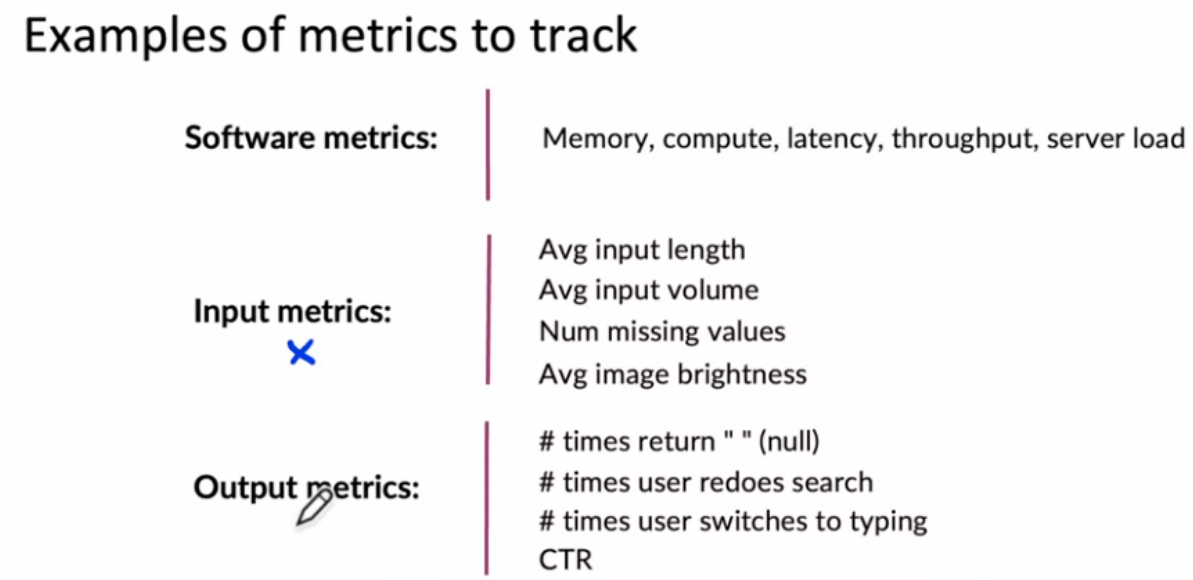
The router changes one deployment or the other



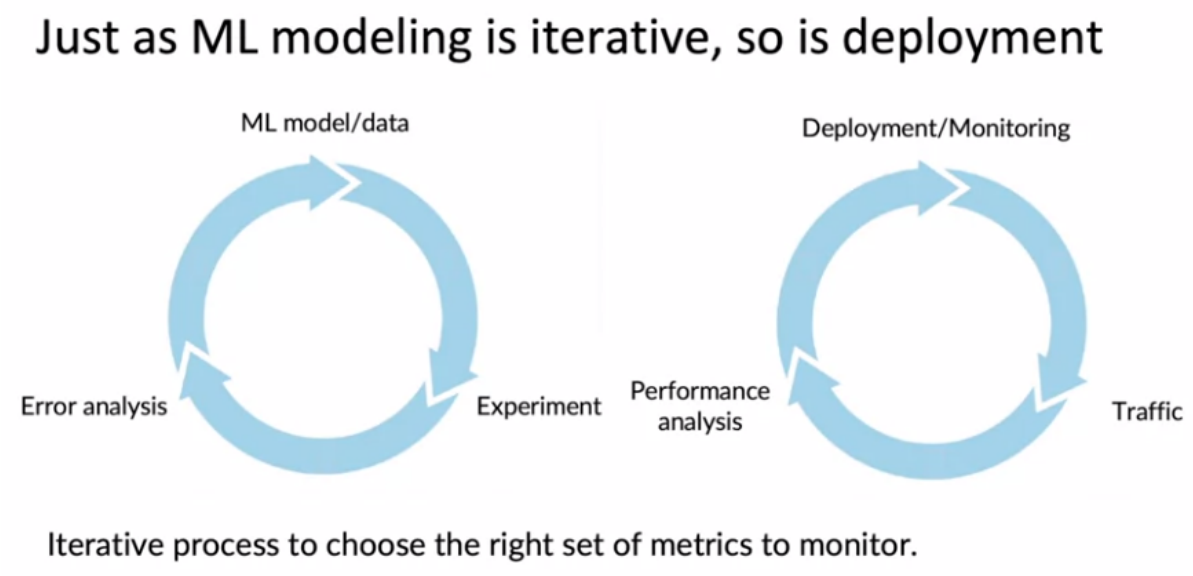
Human in the loop: Steps AI Assistance and Partial Automation



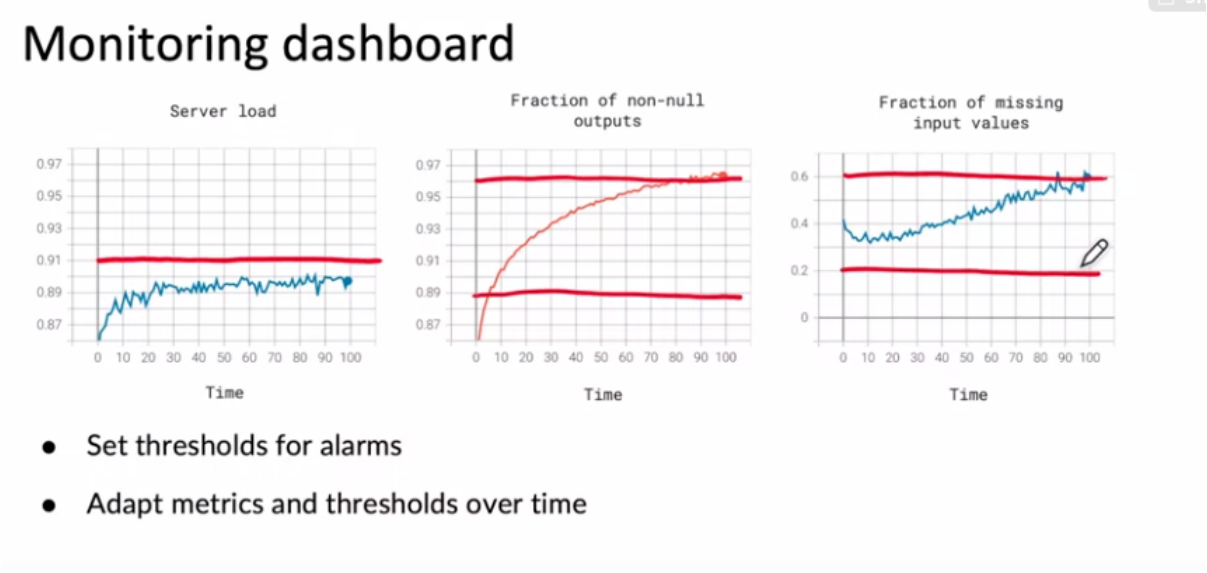
When I'm designing my monitoring dashboards for the first time, I think it's okay to start off with a lot of different metrics and monitor a relatively large set and then gradually remove the ones that you find over time not to be particularly useful.



These output metrics can help you figure out if either your learning algorithm, output y has changed in some way, or if something that comes even after your learning algorithms output, such as the user's switching over to typing has changed in some significant way.

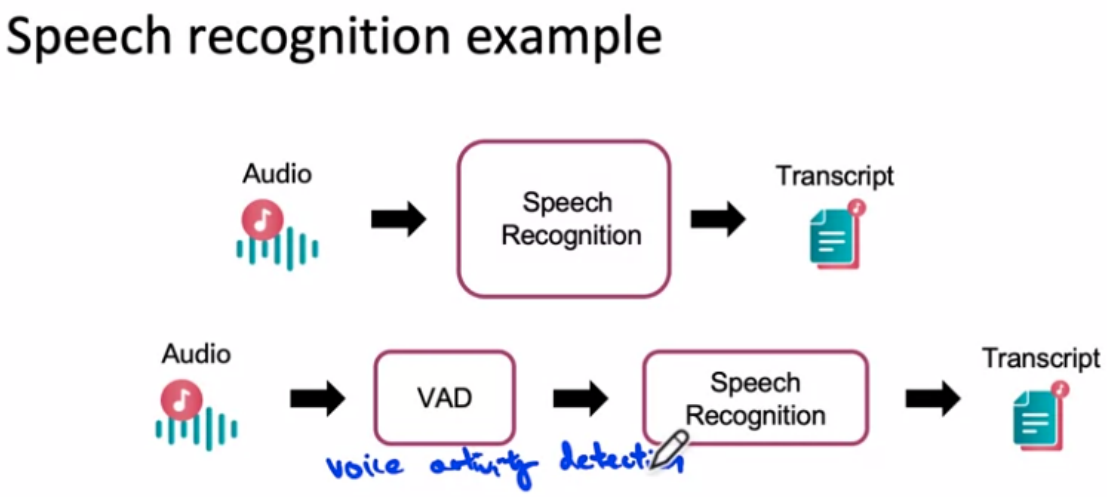


In my experience, it usually takes a few tries to converge to the right set of metrics to monitor.



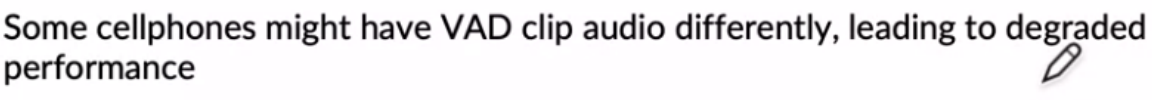
Pipeline Monitoring

Many AI systems are not just a single machine learning model running a prediction service, but instead involves a pipeline of multiple steps. So what are machine learning pipelines and how do you build monitoring systems for that?

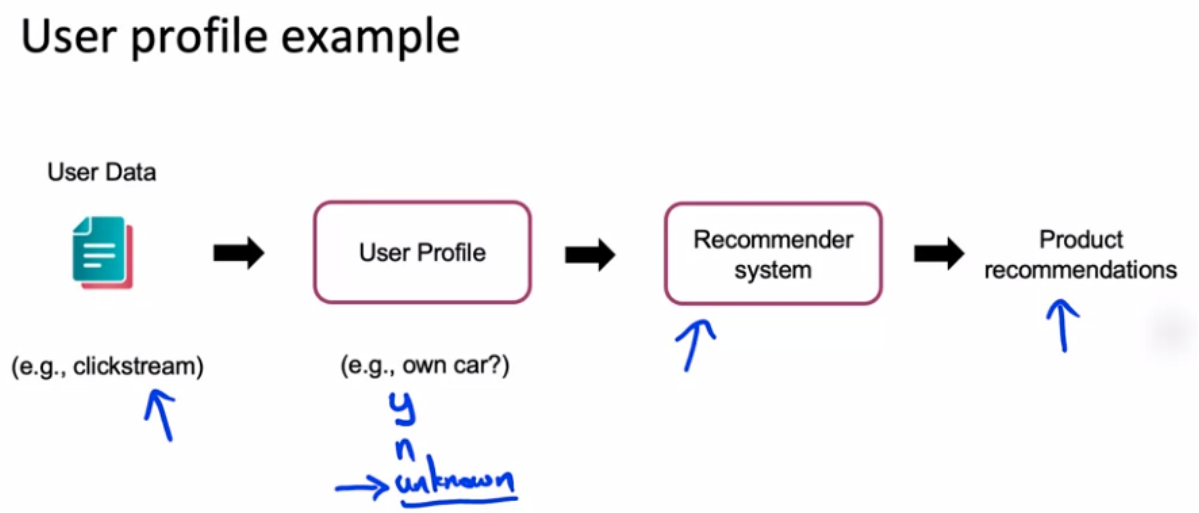


The voice activity detection module looks at the long stream of audio on your cell phone and clips or shortens the audio to just a part where someone is talking and streams only that to the cloud server to perform the speech recognition.

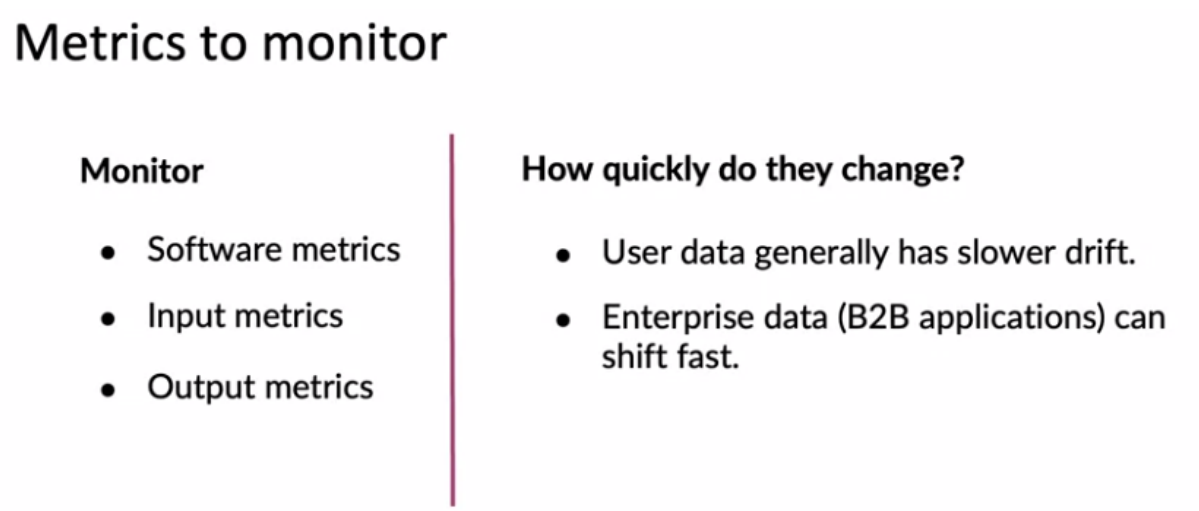
Both VAD and Speech Recognition are ML systems



Maybe it leaves more silence at the start or end or less silence at the start or end and thus if the VAD's output changes, that will cause the speech recognition systems input to change. And that could cause degraded performance of the speech recognition system



When you have a machine learning pipeline, these cascading effects in the pipeline can be complex to keep track of.



Week 1 Optional References

Week 1: Overview of the ML Lifecycle and Deployment

If you wish to dive more deeply into the topics covered this week, feel free to check out these optional references. You won't have to read these to complete this week's practice quizzes.

Concept and Data Drift

Monitoring ML Models

A Chat with Andrew on MLOps: From Model-centric to Data-centric

Papers

Konstantinos, Katsiapis, Karmarkar, A., Altay, A., Zaks, A., Polyzotis, N., … Li, Z. (2020). Towards ML Engineering: A brief history of TensorFlow Extended (TFX). http://arxiv.org/abs/2010.02013

Paleyes, A., Urma, R.-G., & Lawrence, N. D. (2020). Challenges in deploying machine learning: A survey of case studies. http://arxiv.org/abs/2011.09926

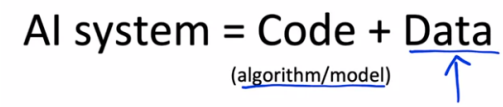
Sculley, D., Holt, G., Golovin, D., Davydov, E., & Phillips, T. (n.d.). Hidden technical debt in machine learning systems. Retrieved April 28, 2021, from Nips.c https://papers.nips.cc/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-Paper.pdf

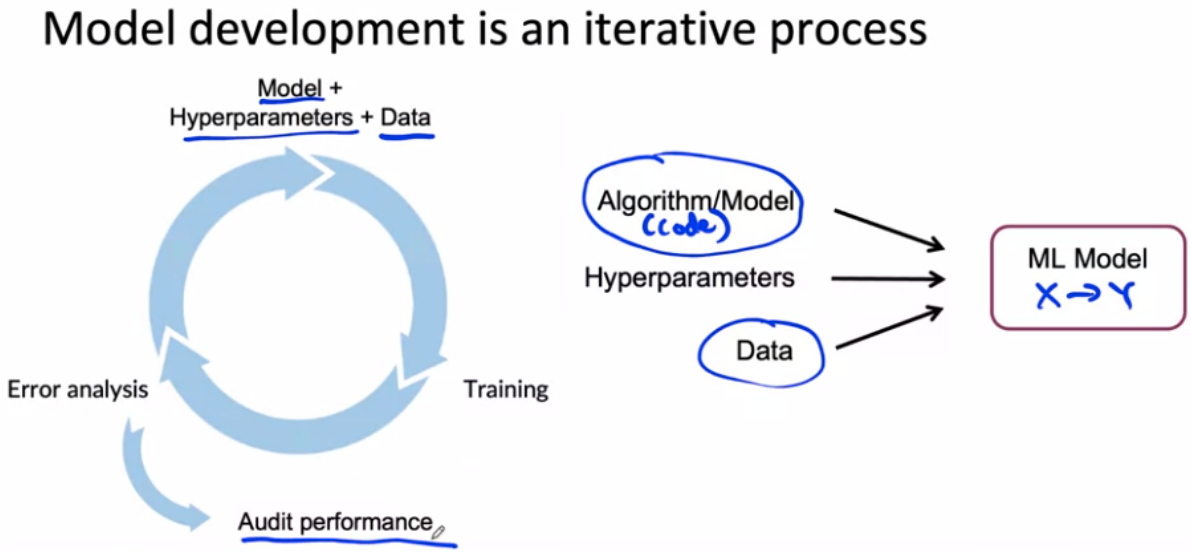
Week 2: Select and Train a Model

This week, our focus will be on the modeling part of the full cycle of a machine learning project, and you learn some suggestions for how to select and train the model, and how to perform error analysis, and use that to drive model improvements.

Selecting and Training a Model

We will fix the model and focus on how we can improve the data in order to obtain better results

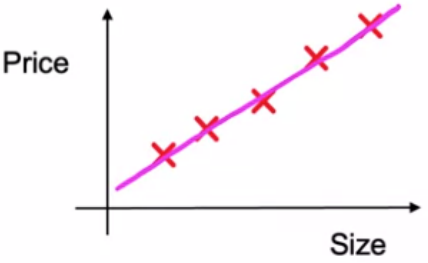




After you've done this enough times and achieve a good model, one last step that's often useful is to carry out a richer error analysis and have your system go through a final audit to make sure that it is working before you push it to a production deployment. So why is model development hard? When building a model, I think there are three key milestones that most projects should aspire to accomplish.

Challenges in model development

1. Doing well on training set (usually measured by average training error).
2. Doing well on dev/test sets.



3. Doing well on business metrics/project goals.

The job of a machine learning engineer would be much simpler if the only thing we ever had to do was do well on the holdout test set. As hard as it is to do well in the holdout test set, unfortunately, sometimes that isn't enough.

Performance on disproportionately important examples

Web Search example

"Apple pie recipe" "Latest movies"
"Wireless data plan" "Diwali festival"
} **Informational and Transactional queries**

"Stanford" "Reddit" "Youtube"
} **Navigational queries**

For informational and transactional queries, a web search engine wants to return the most relevant results, but users are willing to forgive maybe ranking the best result, Number two or Number three. There's a different type of web search query such as Stanford, or Reddit, or YouTube. These are called navigational queries, where the user has a very clear intent, very clear desire to go to Stanford.edu, or Reddit.com, or YouTube.com. When a user has a very clear navigational intent, they will tend to be very unforgiving if a web search engine does anything other than return Stanford.edu as the Number one ranked results and the search engine that doesn't give the right results will quickly lose the trust of its users.

Now one thing you could do is try to give these examples a higher weight. That could work for some applications, but in my experience, just changing the weights of different examples doesn't always solve the entire problem.

Why low average error isn't good enough

Unfortunate conversation in many companies

MLE: "I did well on the test set!"

Product Owner: "But this doesn't work for my application"

MLE: "But... I did well on the test set!"

Performance on key slices of the dataset

Example: ML for loan approval

Make sure not to discriminate by ethnicity, gender, location, language or other protected attributes.

Example: Product recommendations from retailers

Be careful to treat fairly all major user, retailer, and product categories.

Rare classes

Skewed data distribution

99% negative 1% positive

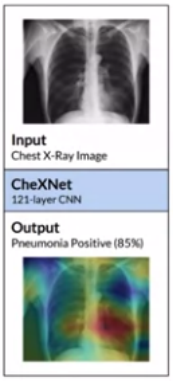
print("0") ←

Accuracy in rare classes

Condition	Performance
Effusion	0.901
Edema	0.924
Mass	0.909
Hernia	0.851

10,000 → Effusion

~100 → Hernia

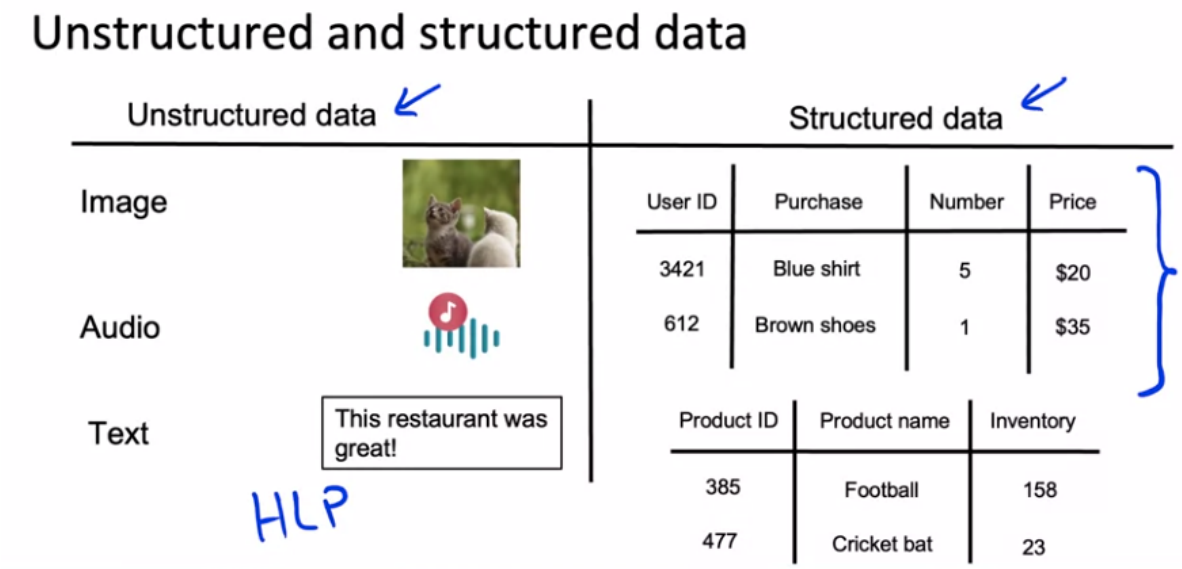


Establishing a baseline level of performance

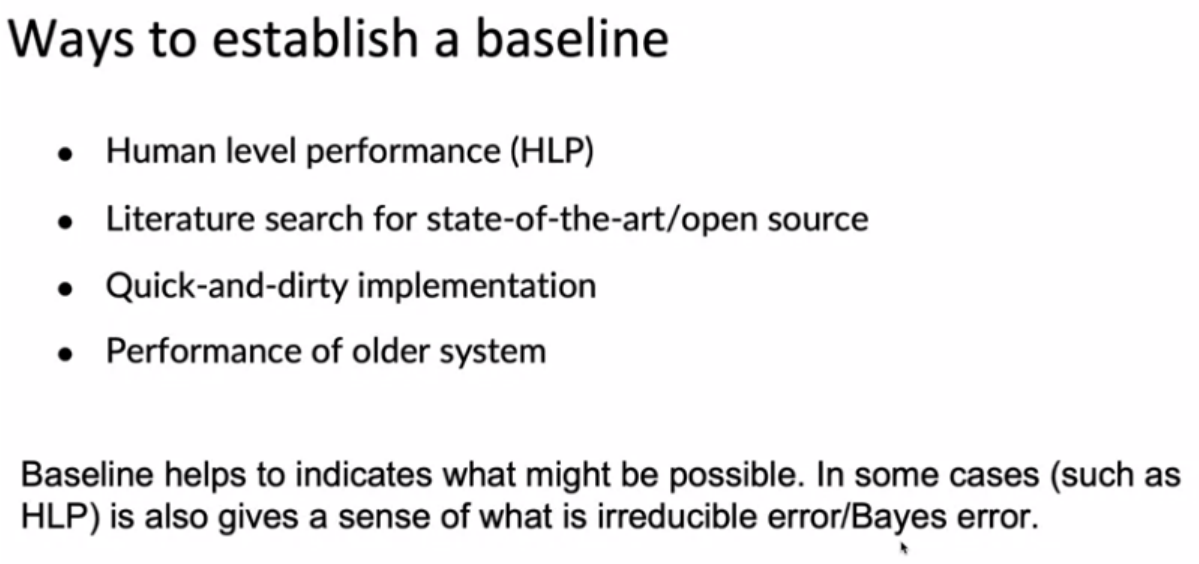
Speech recognition example:

Type	Accuracy	Human level performance	
Clear Speech	94%	95%	1%
→ Car Noise	89%	93%	4%
People Noise	87%	89%	2%
→ Low Bandwidth	70%	70%	~0%

With this analysis, we realized that maybe the low bandwidth audio was so garbled. Even people, humans can't recognize what was said and it may not be that fruitful to work on that. Instead, it may be more fruitful to focus our attention on improving speech recognition with car noise in the background. In this example, using human level performance, which are sometimes abbreviated to HLP, Human Level Performance, gives you a point of comparison or a baseline that helps you decide where to focus your efforts on car noise data rather than on low bandwidth data.



Human level performance (HLP) is generally more effective for establishing a baseline on unstructured data problems (such as images and audio) than structured data problems



We've talked about how machine learning is an iterative process where you start with a model, data, hyperparameters, training model, carry out error analysis, and then use that to drive further improvements. After you've done this a few times, gone around the loop enough times, when you have a good enough model, you might then carry out a final performance audit before taking it to production. In order to get started on this first step of coming of the model, here are some suggestions.

Getting started on modeling

- Literature search to see what's possible (courses, blogs, open-source projects).
- Find open-source implementations if available.
- A reasonable algorithm with good data will often outperform a great algorithm with no so good data.

Don't obsess about taking the algorithm that was just published in some conference last week, that is the most cutting edge algorithm, instead find something reasonable, find a good open source implementation and use that to get going quickly. Because being able to get started on this first step of this loop, can make you more efficient in iterating through more times, and that will help you get to good performance more quickly.

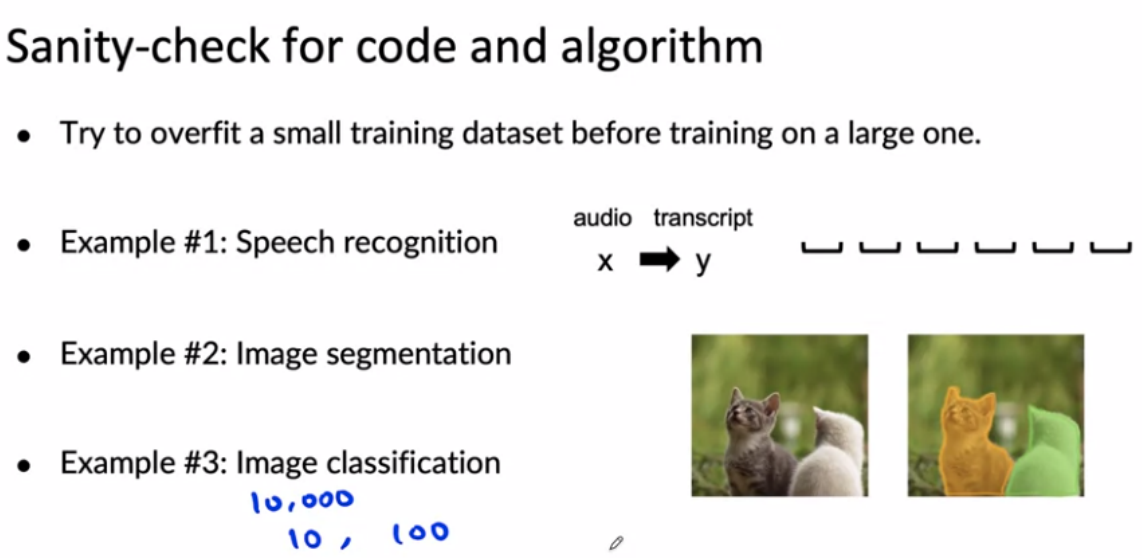
Deployment constraints when picking a model

Should you take into account deployment contraints when picking a model?

Yes, if baseline is already established and goal is to build and deploy.

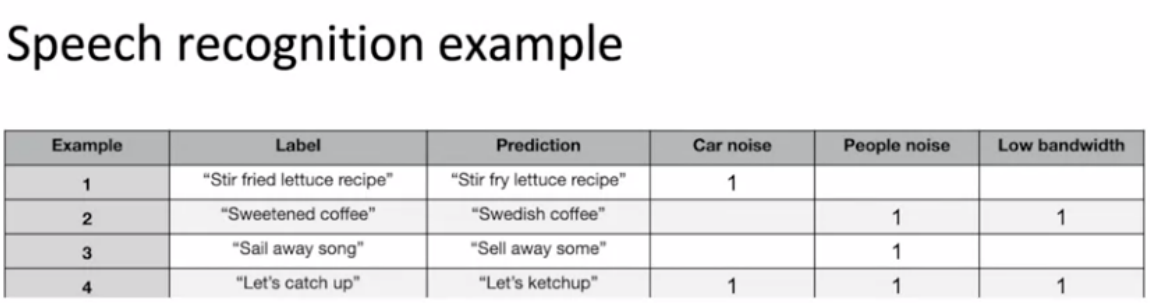
No (or not necessarily), if purpose is to establish a baseline and determine what is possible and might be worth pursuing.

Finally, when trying out a learning algorithm for the first time, before running it on all your data, I would urge you to run a few quick sanity checks for your code and your algorithm. For example, I will usually try to overfit a very small training dataset before spending hours or sometimes even overnight or days training the algorithm on a large dataset. Maybe even try to make sure you can fit one training example, especially, if the output is a complex output. For example, I was once working on a speech recognition system where the goal was to input audio and have a learning algorithm output a transcript. When I trained my algorithm on just one example, one audio clip, when I trained my speech recognition system on just one audio clip on the training set, which is just one audio clip, my system outputs this, it outputs space, space, space, space, space, space. Clearly it wasn't working and because my speech system couldn't even accurately transcribe one training example, there wasn't much point to spending hours and hours training it on a giant training set.



Now, after you've trained a machine learning model, after you've trained your first model, one of the most important things is, how do you carry out error analysis to help you decide how to improve the performance of your algorithm? Let's go on to the next video to dive into error analysis and performance auditing.

Error analysis and performance auditing



Example	Label	Prediction	Car noise	People noise	Low bandwidth
1	"Stir fried lettuce recipe"	"Stir fry lettuce recipe"	1		
2	"Sweetened coffee"	"Swedish coffee"		1	1
3	"Sail away song"	"Sell away some"		1	
4	"Let's catch up"	"Let's ketchup"	1	1	1

Until now, error analysis has typically been done via a manual process, say, in the Jupiter notebook or tracking errors in spreadsheet. I still sometimes do it that way and if that's how you're doing it too, that's fine. But there are also emerging MLOps tools that making this process easier for developers. For example, when my team landing AI works on computer vision applications, the whole team now uses landing lens, which makes this much easier than the spreadsheet.

You've heard me say that training a model is initiative process, deploying a model is an intuitive process. Maybe it should come as no surprise that error analysis is also an iterative process.

Iterative process of error analysis

Visual inspection:

- Specific class labels (scratch, dent, etc.)
- Image properties (blurry, dark background, light background, reflection, ...)
- Other meta-data: phone model, factory

Product recommendations:

- User demographics
- Product features/category

Useful metrics for each tag

- What fraction of errors has that tag? 12%
- Of all data with that tag, what fraction is misclassified? 18%
- What fraction of all the data has that tag?
- How much room for improvement is there on data with that tag?

So by brainstorming different tags, you can segment your data into different categories and then use questions like these to try to decide what to prioritize working on.

Prioritizing what to work on

Type	Accuracy	Human level performance	Gap to HLP	% of data	
Clean Speech	94%	95%	1%	60%	→ 0.6%
Car Noise	89%	93%	4%	4%	→ 0.16%
People Noise	87%	89%	2%	30%	→ 0.6%
Low Bandwidth	70%	70%	0%	6%	→ ~0%

And so whereas previously we had said there's a lot of room for improvement in car noise, in this slightly richer analysis, we see that because people noise accounts for such a large fraction of the data, it may be more worthwhile to work on either people noise or maybe on clean speech because there's actually larger potential for improvements in both of those than for speech with car noise. So to summarize, when prioritizing what to work on, you might decide on the most

important categories to work on based on, how much room for improvement there is, such as compared to human level performance or according to some baseline comparison.

Prioritizing what to work on

Decide on most important categories to work on based on:

- How much room for improvement there is.
- How frequently that category appears.
- How easy is to improve accuracy in that category.
- How important it is to improve in that category.

Adding/improving data for specific categories

For categories you want to prioritize:

- Collect more data
- Use data augmentation to get more data
- Improve label accuracy/data quality

Data sets where the ratio of positive to negative examples is very far from 50-50 are called skewed data sets. Let's look at some special techniques for handling them.

Examples of skewed datasets

Manufacturing example

99.7% no defect $y=0$

0.3% defect $y=1$

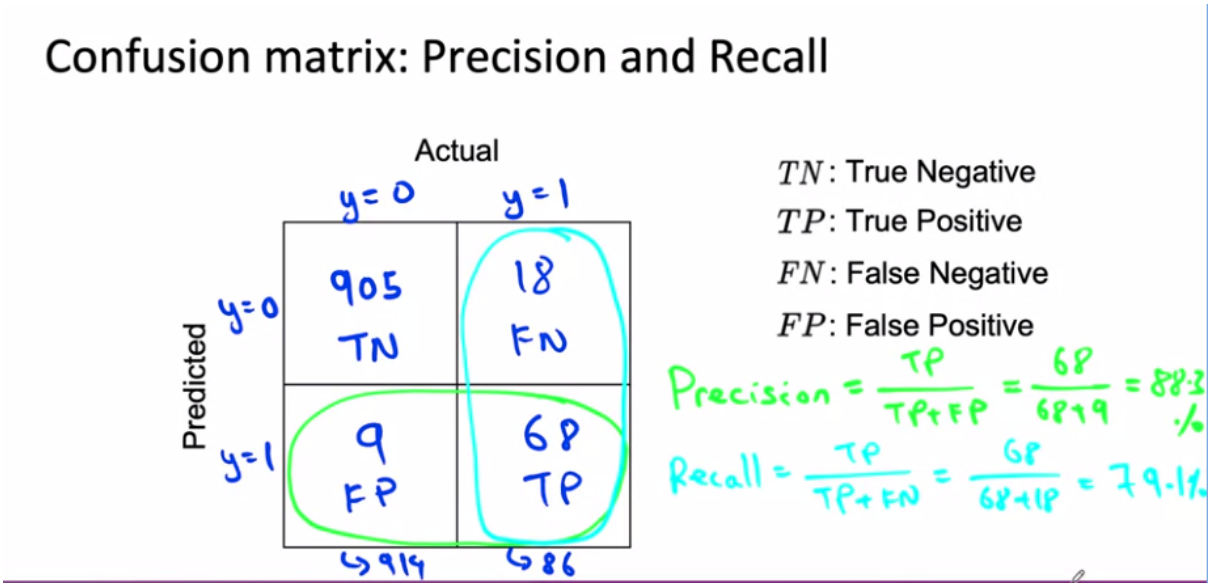
print("0")

99.7%

Medical Diagnosis example: 99% of patients don't have a disease

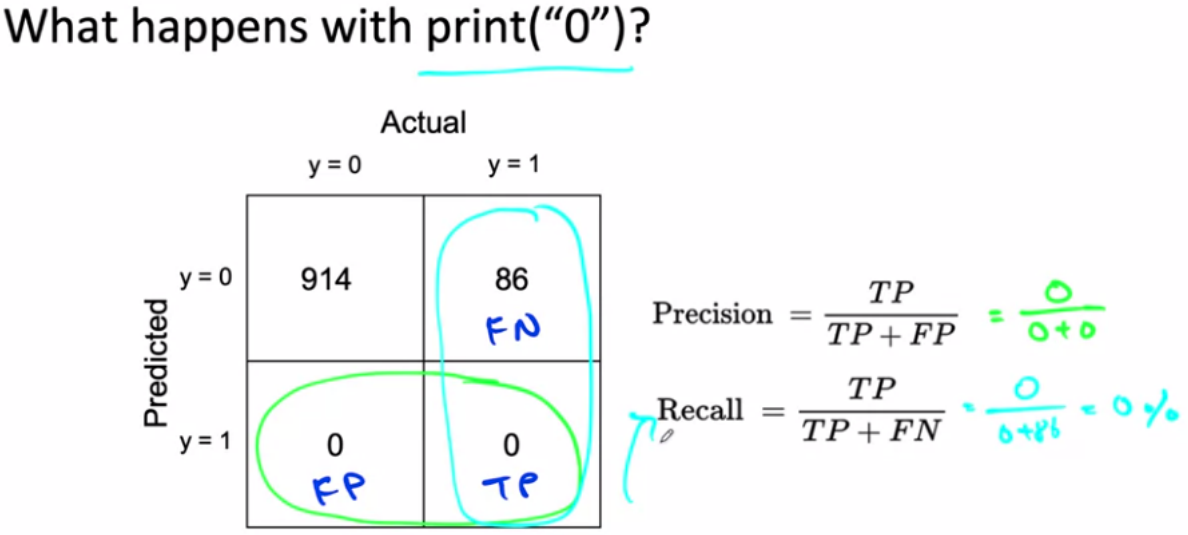
Speech Recognition example: In wake word detection, 96.7% of the time wake word doesn't occur

When you have a very skewed data set like this, raw accuracy is not that useful a metric to look at because Prince Zero can get very high accuracy. Instead, it's more useful to build something called the confusion matrix. A confusion matrix is a matrix where one axis is labeled with the actual label, is the ground truth label, y equals 0 or y equals 1 and whose other axis is labeled with the prediction.



This is indeed a pretty skewed data set where out of 1000 examples there were 940 negative examples and just 86 positive examples, 8.6 percent positive, 91.4 percent negative.

Let's see what happens if your learning algorithm outputs zero all the time. It turns out it won't do very well on recall.



Combining precision and recall – F_1 score

	Precision (P)	Recall (R)	F_1
Model 1	88.3	79.1	83.4% ←
Model 2	97.0	7.3	13.6%

$$F_1 = \frac{2}{\frac{1}{P} + \frac{1}{R}}$$

Being F1 the harmonic mean of Precision and recall, the score punished the model if one of those metrics gave a low score.

So far, we've talked about the binary classification problem with skewed data sets. It turns out to also frequently be useful for multi-class classification problems.

Multi-class metrics

Classes: Scratch, Dent, Pit mark, Discoloration

Defect Type	Precision	Recall	F_1
Scratch	82.1%	99.2%	89.8%
Dent	92.1%	99.5%	95.7%
Pit mark	85.3%	98.7%	91.5%
Discoloration	72.1%	97%	82.7%

By combining precision and recall using F_1 as follows, this gives you a single number evaluation metric for how well your algorithm is doing on the four different types of defects and can also help you benchmark to human-level performance and also prioritize what to work on next. Instead of accuracy on scratches, dents, pit marks, and discolorations, using F_1 score can help you to prioritize the most fruitful type of defect to try to work on. The reason we use F_1 is

because, maybe all four defects are very rare and so accuracy would be very high even if the algorithm was missing a lot of these defects. I hope that these tools will help you both evaluate your algorithm as well as prioritize what to work on, both in problems with skewed data sets and for problems with multiple rare classes.

Performance auditing

Even when you're learning algorithm is doing well On accuracy or F one score or some appropriate metric is often worth one last performance audit before you push it to production. And this can sometimes save you from significant post deployment problems.

Auditing framework

Check for accuracy, fairness/bias, and other problems.

1. Brainstorm the ways the system might go wrong.
   - Performance on subsets of data (e.g., ethnicity, gender).
   - How common are certain errors (e.g., FP, FN).
   - Performance on rare classes.
2. Establish metrics to assess performance against these issues on appropriate slices of data.
3. Get business/product owner buy-in.

Speech recognition example

1. Brainstorm the ways the system might go wrong.
   - Accuracy on different genders and ethnicities.
   - Accuracy on different devices.
   - Prevalence of rude mis-transcriptions.

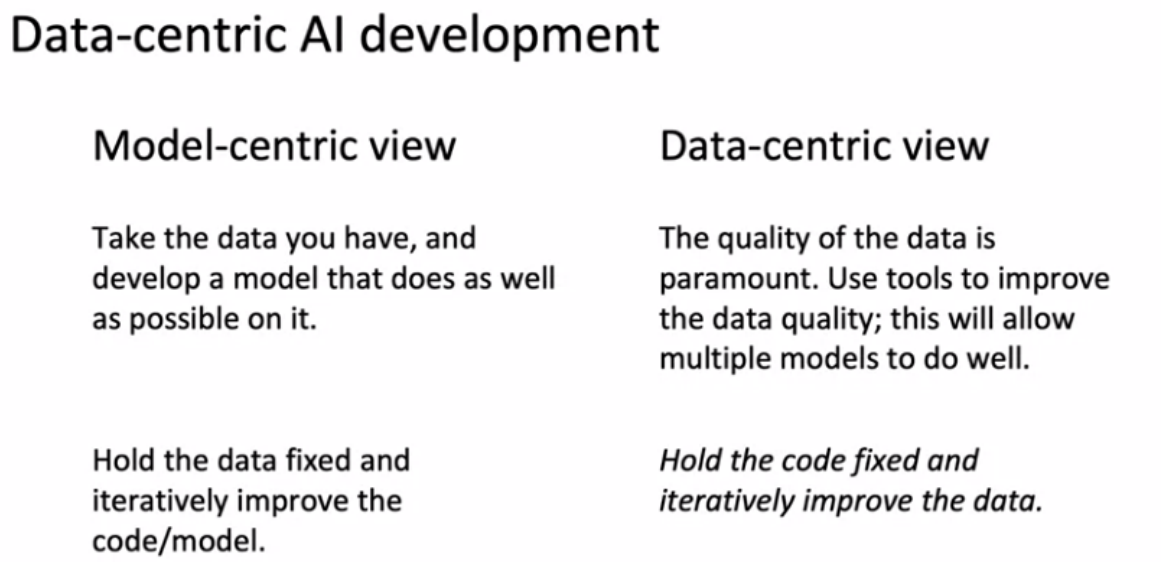
GAN gun gang

2. Establish metrics to assess performance against these issues on appropriate slices of data.
   - Mean accuracy for different genders and major accents.
   - Mean accuracy on different devices.
   - Check for prevalence of offensive words in the output.

Data iteration

With a model centric view of AI developments, you would take the data you have and then try to work really hard to develop a model that does as well as possible on the data because a lot of academic research on AI was driven by researchers, downloading a benchmark data set and trying to do well on that benchmark. Most academic research on AI is model centric, because the benchmark data set is a fixed quantity. In this view, model centric development, you would hold the data fixed and iterative the improve. In this model centric view, you would hold the data fixed and iteratively improve the code or the model.

Which is to shift a data from a model centric to what a data centric view. In this view, we think of the quality of the data as paramount, and you can use tools such as era analysis or data augmentation to systematically improve the data quality. For many applications, I find that if your data is good enough, there are multiple models that will do just fine. In this view, you can instead hold the code fixed and iteratively improve the data.



If you've been used to model centric thinking for most of your experience with machine learning, I would urge you to consider taking a data centric view as well, where when you're trying to improve your learning outcomes performance, try asking how can you make your data set even better?

A useful picture of data augmentation

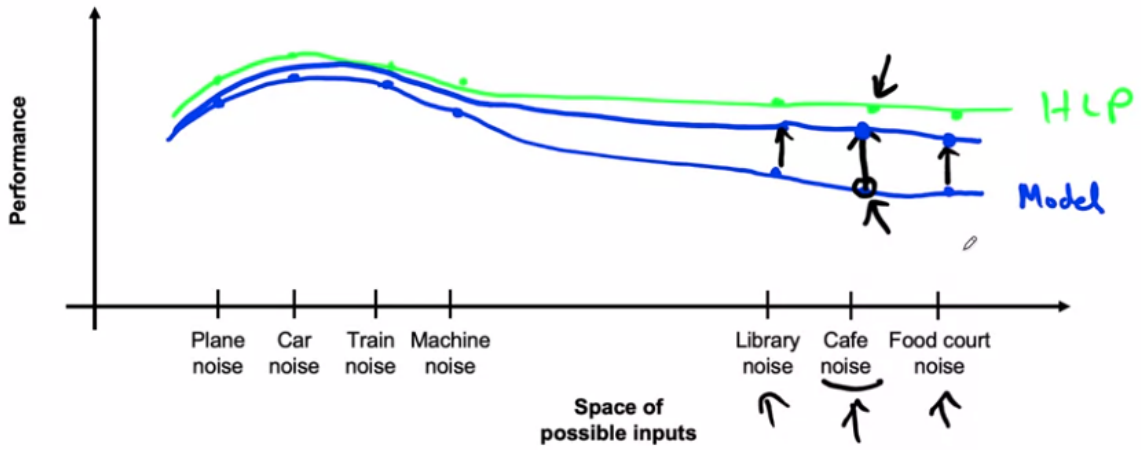
Speech recognition example

Different types of speech input:

- Car noise
- Plane noise
- Train noise
- Machine noise
- Cafe noise
- Library noise
- Food court noise

There are two types of noise: ones that come with mechanical noise and ones with human noise.

Speech recognition example



So this gap represents an opportunity for improvement. Now, what happens if you use data augmentation or maybe not data augmentation but go out to a bunch of actual cafes, to collect a lot more data with cafe noise in the background. What you'll do is, you'll take this point imagine grabbing a hold of this blue rubber bands or this rubber sheet, and pulling it upward like so. That's what you're doing if you collect or somehow gets more data with cafe noise and add that your training set, you're pulling up the performance of the algorithm on inputs with cafe noise. And what that will tend to do, is pull up this rubber sheet in the adjacent region as well. So if performance on cafe noise goes up, probably performance on the nearby points will go up to and performance on far away. Points may or may not go up as much. It turns out that for unstructured data problems, pulling up one piece of this rubber sheet is unlikely to cause a different piece of the rubber sheet to dip down really far below. Instead, pulling up one point causes nearby points to be pulled up quite a lot and far away points may be pulled up a little bit, or if you're lucky, maybe more than a little bit.

Data augmentation

Goal:

Create realistic examples that (i) the algorithm does poorly on, but (ii) humans (or other baseline) do well on

Checklist:

- Does it sound realistic?
- Is the x → y mapping clear? (e.g., can humans recognize speech?)
- Is the algorithm currently doing poorly on it?

Let's say that you have a very small set of images of smartphones with scratches. Here's how you may be able to use data augmentation. You can take the image and flip it horizontally. This results in a pretty realistic image. The phone buttons are now on the other side, but this could be a useful example to add to your training set. Or you could implement contrast changes or actually brighten up the image here so the scratch is a little bit more visible. Or you could try darkening the image, but in this example, the image is now so dark that even I as a person can't really tell if there's a scratch there or not. Whereas these two examples on top would pass the checklist we had earlier, that the human can still detect the scratch well, this example is too dark, it would fail that checklists. I would try to choose the data augmentation scheme that generates more examples that look like the ones on top and fill the ones that look like the ones here at the bottom.

If you're using data augmentation, you're adding to specific parts of the training set such as adding lots of data with cafe noise. So now you're training set may come from a very different distribution than the death set and the test set. Is this going to hurt your learning album's performance?

Can adding data hurt performance?

For unstructured data problems, if:

- The model is large (low bias).
- The mapping x → y is clear (e.g., given only the input x, humans can make accurate predictions).

Then, **adding data rarely hurts accuracy.**

Photo OCR counterexample

Adding a lot of new "I"s may skew the dataset and hurt performance

If we want to identify the "I"s we can use data augmentation, but by adding new examples the third option can be considered an "I" when, as being ambiguous, it should be considered a 1.

For many structure data problems. It turns out that creating brand new training examples is difficult, but there's something else you could do which is to take existing training examples and figure out if there are additional useful features you can add to it.

Structured data

Restaurant recommendation example

Vegetarians are frequently recommended restaurants with only meat options.

Possible features to add?

- Is person vegetarian (based on past orders)?
- Does restaurant have vegetarian options (based on menu)?

Additional features like these, can be hand coded or they could in turn be generated by some learning algorithm, such as having a learning average home, try to read the menu and classify meals as vegetarian or not, or having people called this manually could also work depending on your application.

Other food delivery examples

- Only tea/coffee
- Only pizza

What are the added features that can help make a decision?

Product recommendation:

Collaborative filtering → Content based filtering Cold-start

Over the last several years, there's been a trend in product recommendations of a shift from collaborative filtering approaches to what content based filtering approaches. Collaborative filtering approaches is loosely an approach that looks at the user, tries to figure out who is similar to that user and then recommends things to you that people like you also liked. In contrast, a content based filtering approach will tend to look at you as a person and look at the description of the restaurant or look at the menu of the restaurants and look at other information about the restaurant, to see if that restaurant is a good match for you or not. The advantage of content based filtering is that even if there's a new restaurant or a new product that hardly anyone else has liked by actually looking at the description of the restaurant, rather than just looking at who else like the restaurants, you can more quickly make good recommendations. This is sometimes also called the Cold Start Problem. How do you recommend a brand new product that almost no one else has purchased or like or dislike so far? And one of the ways to do that is to make sure that you capture good features for the things that you might want to recommend. Unlike collaborative filtering, which requires a bunch of people to look at the product and decide if they like it or not, before it can decide whether a new user should be recommended the same product.

- Error analysis can be harder if there is no good baseline (such as HLP) to compare to.
- Error analysis, user feedback and benchmarking to competitors can all provide inspiration for features to add.

As you're working to iteratively improve your algorithm. One thing, that'll help you be a bit more efficient is to make sure that you have robust experiment tracking.

Experiment tracking

What to track?

- Algorithm/code versioning
- Dataset used
- Hyperparameters
- Results

Tracking tools

- Text files
- Spreadsheet
- Experiment tracking system

Desirable features

- Information needed to replicate results
- Experiment results, ideally with summary metrics/analysis
- Perhaps also: Resource monitoring, visualization, model error analysis

Rather than worrying too much about exactly which experiment tracking framework to use though, the number one thing I hope you take away from this video is, do got to have some system, even if it's just a text file or just a spreadsheet for keeping track of your experiments and include as much information as is convenient to include. Because later on, if you try to look back, remember how you had generated a certain model, having that information would be really useful for helping you to replicate your own results.

I'd like to leave you with a thought on shifting from big data to good data.

From Big Data to Good Data

Try to ensure consistently high-quality data in all phases of the ML project lifecycle.

Good data:

- Covers important cases (good coverage of inputs x) ←
- Is defined consistently (definition of labels y is unambiguous)
- Has timely feedback from production data (distribution covers data drift and concept drift)
- Is sized appropriately

Week 2 Optional References

Week 2: Select and Train Model

If you wish to dive more deeply into the topics covered this week, feel free to check out these optional references. You won't have to read these to complete this week's practice quizzes.

Establishing a baseline

Error analysis

Experiment tracking

Papers

Brundage, M., Avin, S., Wang, J., Belfield, H., Krueger, G., Hadfield, G., … Anderljung, M. (n.d.). Toward trustworthy AI development: Mechanisms for supporting verifiable claims∗. Retrieved May 7, 2021http://arxiv.org/abs/2004.07213v2

Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B., & Sutskever, I. (2019). Deep double descent: Where bigger models and more data hurt. Retrieved from http://arxiv.org/abs/1912.02292

Week 3: Data Definition and Baseline

Define Data and Establish Baseline

Label and Organize Data

Scoping (optional)